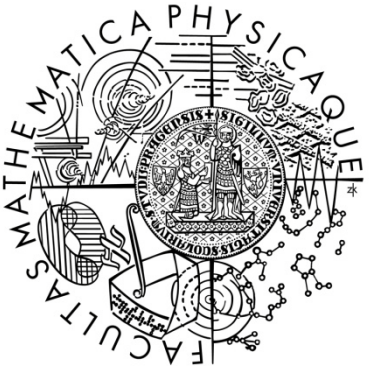
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tuan Do Manh

Nástroj pro porovnání různých metod řešení nelineárních rovnic

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Tomáš Mikula

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha, 2012

Poděkování

Chtěl bych poděkovat zejména dvěma pánům, bez kterých by nebylo možné zrealizovat tuto práci. Nejprve děkuji svému vedoucímu, panu Mgr. Tomáši Mikulovi, který mě v tomto projektu vedl už od samého začátku, a i přes svoji nepřítomnost zde v České republice mě nepřestal podporovat. Děkuji mu za jeho rady, připomínky, kritiku, které po malých kouskách vylepšovali moji práci, a především za jeho čas, který mi věnoval. V další řadě bych chtěl poděkovat panu prof. RNDr. Vítovi Dolejšímu, Ph.D., DSc., který mě přivedl k tématu této bakalářské práce. Děkuji mu za jeho čas a za jeho konzultace, pomocí kterých jsem dokázal hlouběji nahlédnout do matematického problému. V poslední řadě bych chtěl poděkovat všem vyučujícím na škole, od kterých jsem měl možnost se učit. To díky nim jsem získal vědomosti, které jsem dokázal v této práci zůročit.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne..... podpis

Název práce: Nástroj pro porovnání různých metod řešení nelineárních rovnic

Autor: Tuan Do Manh

Katedra / Ústav: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Tomáš Mikula, Katedra aplikované matematiky

Abstrakt: Cílem této práce je vytvořit nástroj pro řešení nelineárních rovnic numerickými metodami. Využívá při tom jak pomalých metod pro hledání kořene jako metoda bisekce, metoda regula falsi, tak i rychle počítajících metod jako je Newtonova metoda. Newtonova metoda, i když obecně rychlá metoda, dokáže být v jistých případech velmi problematická. Nekonverguje vždy k řešení rovnice. Proto v této práci implementuji modifikované metody, které se snaží nějak se vypořádat s nedostatkem Newtonovy metody. Program má posloužit jako nástroj pro porovnávání a vyhodnocování efektivnosti výpočtů jednotlivých metod v různých situacích.

Klíčová slova: nelineární rovnice, numerické metody, Newtonova metoda

Title: A tool for evaluation of different methods for solving nonlinear equations

Author: Tuan Do Manh

Department: Department of Applied Mathematics

Supervisor: Mgr. Tomáš Mikula, Department of Applied Mathematics

Abstract: The objective of this work is to create a tool for solving nonlinear equations using numeric methods. It uses both slow working methods, such as bisection method or regula falsi method, and fast working methods, such as Newton's method. The Newton's method, while fast, can be very problematic in certain scenarios. It does not always converse to the root of the equation. That is why in this work, I try to implement modified methods, which attempt to deal with the imperfections of the Newton's method. The program is suppose to be a good tool for comparing and evaluating the efficiency of each methods in different situations.

Keywords: nonlinear equations, numerical methods, Newton's method

Obsah

Úvod / Předmluva

Obecný úvod do problematiky

V matematických a technických oborech, ale i v běžném životě, se setkáváme s různými druhy matematických rovnic. Člověk sám o sobě však dokáže řešit pouze malé množství z té velké škály rovnic, se kterými se setkává v technických oborech, matematických oborech nebo v normálním životě. Uvážíme-li na chvíli například jen polynomiální rovnice. Dokážeme řešit lineární, kvadratické, kubické i bikvadratické rovnice. Ale kdybychom se dostali k řešení polynomiální rovnici řádu 5, už bychom takovou úlohu řešit pomocí obecného vzorce nedokázali. Toto tvrzení, že neexistuje obecný vzorec pro výpočet kořenů rovnice řádu 5 a rovnic vyšších řádů, se dokazuje v algebře. V takových případech hledáme východisko v numerické matematice, oboru matematiky, která se zabývá řešením matematických problémů pomocí počítače.

V numerické matematice existují metody a algoritmy, pomocí kterých dokážeme za určitých předpokladů a za dodání vstupních počátečních parametrů počítat přibližná řešení rovnic. Všechny tyto metody jsou založené na tom, že se snažíme iteračně počítat a upravovat naše přibližné řešení tak dlouho, dokud nenajdeme přesně kořen rovnice, který jsme chtěli najít, nebo alespoň dostat naše přibližné řešení tak blízko k hledanému řešení, aby splňovalo nějakou zadanou přesnost.

Mohli bychom například jmenovat metodu bisekce, která dostane jako vstupní parametr interval [a, b] a přesnost, se kterou má počítat. Podle algoritmu se pak zadaný interval půlí a zmenšuje, dokud se nepřiblížíme dostatečně blízko k hledanému kořenu. Další metodou je metoda sečen (neboli metoda regula falsi), která nepůlí vstupní interval, ale dělí ho pomocí sečné přímky. Obě tyto zmíněné metody patří mezi tzv. vždy konvergentní metody. Jak již název napovídá, vždy konvergují k hledanému řešení, ať už to je pomalu nebo rychle. Poslední numerickou metodu pro řešení lineárních, ale především nelineárních rovnic, je Newtonova metoda, neboli metoda tečen. Ta dostane jako vstupní parametr přesnost a počáteční bod x_0, ze kterého pak povede tečnou přímku pro upravení a zpřesnění počátečního přibližného kořene. Newtonova metoda v průměrném případě funguje při splnění určitých podmínek dobře. Avšak pro Newtonovu metodu bychom dokázali vymyslet

případ, kdy se početní postup nebude chovat hezky a i přes dlouhé počítání bychom k toužebnému, hledanému kořenu nedospěli. Newtonova metoda totiž není vždy konvergentní.

Většina matematického softwaru dnes dostupná poskytuje základní implementované metody pro řešení lineárních i nelineárních rovnic. Pro Newtonovu metodu je třeba splnit jisté podmínky. Pak můžeme jistě říct, že výpočet bude správně konvergovat k hledanému řešení. Ale co když tyto podmínky splněny nejsou? Jestli bude výpočet konvergovat a my dospějeme k nějakému řešení nebo ne, to předem nevíme. Newtonova metoda má různé modifikace a verze, které se snaží nějak se vypořádat s nedostatkem ne vždy konvergentnosti.

Cílem této bakalářské práce není vymyslet novou převratnou metodu, pomocí které by se dal řešit problém nelineárních rovnic. V numerické matematice je tento problém již dlouho podrobně zkoumán a prostudován. Existují i nástroje, které nelineární rovnice řeší. Kladu si však za cíl vytvořit nástroj, který implementuje modifikované rozšiřující metody k základním metodám. Také by to měl být nástroj, pomocí kterého bychom mohli porovnávat a zkoumat jaké metody v jakém případě pracují lépe a hůře.

Stručný popis bakalářské práce

Na začátek této bakalářské práce nejdříve se podíváme na některé existující programy a software, který je k dostání. Rozebereme, co umí a co nabízejí. Vytkneme body, ve kterých se náš program bude lišit, a to, co bude náš program umět navíc a jinak.

V další kapitole představíme matematický problém, kterým se budeme zabývat. Zavedeme si nějaké matematické pojmy a pár definicí týkající se nelineárních rovnic a metod spojených s výpočtem a hledáním řešení rovnic. Taktéž si obecně zformulujeme úlohu hledání kořene rovnice z hlediska numerické matematiky. Poukážeme na nedostatky jednotlivých metod a navrhneme nějaká možná řešení.

Další částí této bakalářské práce se podíváme pod pokličku programu. Budeme rozebírat program z nejsvrchnější vrstvy a postupně budeme pronikat detailněji do vnitřních vrstev. Podíváme se na jednotlivé části programu a řekneme si, jak a proč byly takhle věci implementovány. A celkově popíšeme, co s čím pracuje a jak všechny části programu do sebe zapadají.

Dále si představíme program z hlediska uživatele. Bude zde popsáno, jak s programem zacházet a jak ho používat. Řekneme si rovněž pár rad a doporučení k postupu při počítání s nelineárními rovnicemi.

V závěru bakalářské práce zrekapitulujeme naše kladené cíle. Shrneme, co se nám povedlo zrealizovat a zhodnotíme naše úsilí.

1. Přehled existujících programů a software

V této první kapitole se podíváme na některé z dostupných matematických programu a software, který je dnes dostupný. Bude řeč o velikánech jako Wolfram Mathematica [1], Maple [6] nebo Matlab [7]. Všechny tyto programy dokážou řešit rovnice numerickými metodami. Podíváme se trochu podrobněji, co který program nabízí a poukážeme na to, v čem se náš program bude lišit, co umí navíc a jinak.

1.2. Wolfam Mathematica

Program Mathematica od společnosti Wolfram je velmi obsáhlý a kompletní balík pro řešení matematických věcí. Má velmi mnoho příkazů a funkcí, které řeší celou spoustu problému. Co se týče funkcí a rovnic, umí kreslit funkce a grafy funkcí například příkazem `Plot`. Pro řešení rovnic numerickými metodami je zde funkce nesoucí jméno `FindRoot`.

Základní forma příkazu `FindRoot` může vypadat například takto:

```
FindRoot[Sin[x] + Exp[x] == 0, {x, 0}]
```

Tento příkaz se pokusí vypočítat kořen rovnice $\sin(x) + \exp(x) = 0$. Jako vstupní parametr dostává počáteční kořen $x_0 = 0$. Tento počáteční kořen je nutný pro řešení rovnice Newtonovou metodou, která je implicitně nastavena jako výchozí metoda pro příkaz `FindRoot`. Mathematica tedy nabízí řešení rovnic pomocí nějaké modifikované Newtonovy metody. Mimo to máme možnost nastavit řešení metodou regula falsi, kterou se budeme zabývat v kapitole 2.2.3., nebo ještě Brentovou metodou [8].

Funkci `FindRoot` se dají pak nastavit další parametry, aby třeba vypsala počet provedených iterací. Dá se ji nastavit maximální počet iterací, který se nesmí překročit. Mimo jiné může uživatel také nastavit parametr `DampingFactor`, který slouží jako urychlovač výpočtu hrubou silou. Tento damping faktor úzce souvisí s damping koeficientem, který si představíme v kapitole 2.3.2. spolu s Damping Newtonovou metodou, ale způsob našeho využití damping koeficientu je zcela jiný.

Pro více informací o programu Mathematica, nechť se čtenář odkáže na knihy [9] a [10].

1.2. Maple

Program Maple je další hodně známý matematický program. Obdobně jako Wolfram Mathematica nabízí celou škálu funkcí a možností. Pro řešení rovnic má Maple funkci `fsolve`.

Například bychom mohli v Maplu zadat následující příkaz:

```
fsolve(sin(x) + exp(x))
```

Jak si může čtenář všimnout, není mu třeba zadávat žádné další vstupní parametry. Funkce `fsolve` má implicitně nastavený nějaký interval, ve kterém automaticky hledá, a pravděpodobně když v dané oblasti žádný kořen nenajde, přemístí se jinam a zkouší to jinde. Pokud ale chceme upřesnit, v jakém intervalu chceme hledat, nebo zadat přibližné řešení, tak je to možné.

Další zajímavou možností u funkce `fsolve`, která stojí za povšimnutí, je parametr `avoid`. Funkci `fsolve` se zadá množina bodů, která se má při výpočtu přeskočit. Tato možnost se například využije, když počítáme s nespojitou funkcí a chceme se vyvarovat některých bodům.

Poslední věc v Maplu, která stojí za zmínku, je funkce `NewtonsMethod` ve studentském balíčku *Calculus1*. Je to funkce, která provádí 5 základních iterací Newtonovy metody. Je možné si vyžádat přibližné kořeny v těch pěti iteracích nebo i obrázek průběhu výpočtu nebo dokonce i animaci. Tato funkce je velmi názorná pro vysvětlení Newtonovy metody.

Co však Maple nenabízí, je možnost zvolit si metodu pro řešení. V Maplu jsou naimplementované nějaké numerické metody pro řešení rovnic, ale jsou pro uživatele zcela nepřístupné. Navíc funkce `fsolve` nevrací počet iterací, které funkce provedla při výpočtu.

Pokud zajímá čtenáře program Maple více, odkažme ho na blog [11] Ing. V. Žáka.

1.3. Matlab

Matlab je další velmi využívaný matematický software. Jedná se o interaktivní programové prostředí a skriptovací jazyk čtvrté generace.

V Matlabu najdeme funkci `solve`. Tato funkce není primárně určena pro numerické řešení rovnic, ale snaží se rovnici vyřešit symbolicky. Pokud funkce `solve` neuspěje v nalezení symbolického řešení, pak se pokusí rovnici vyřešit numericky. Funkce `solve` předpokládá polynomiální rovnici. Pokud ji dostane, pokusí se najít všechny kořeny. Jestli dostane transcendentní rovnici, pak se pokusí najít jen jeden kořen (jak je uvedeno ve webové dokumentaci Matlabu [7]).

Další funkci, kterou Matlab nabízí, je funkce `roots`. Tato funkce umí jen najít kořeny polynomu, který je jí zadán ve formě vektoru.

Jak zde vidíme, Matlab nenabízí moc možností pro řešení nelineárních rovnic. Pokud bychom chtěli pracovat s nepolynomiálními rovnicemi, tak by to v Matlabu mohl být docela problém, jelikož vestavěné funkce Matlabu jsou velmi omezující. Jak už jsme ale zmínili, Matlab je i skriptovací jazyk. Takže uživatel má možnost si napsat svoji vlastní funkci pro řešení nelineárních rovnic libovolnou metodou. To však vyžaduje dobrou znalost skriptovacího jazyka a pochopení samotné numerické metody.

Pokud má čtenář hlubší zájem o numerické metody v rámci Matlabu, nechť se odkáže na knihu [12].

1.4. Shrnutí

Matlab nám pro řešení nelineárních rovnic nabízí nejméně. Dokáže hlavně efektivně počítat jen polynomiální rovnice. Uživatel však má možnost si sám naprogramovat svoji vlastní funkci pro řešení rovnic. Maple nabízí kvalitní řešení nelineárních rovnic, kde je většina věcí již automaticky nastavena. Na jednu stranu je to velmi pohodlné, ale kdybychom chtěli studovat různé numerické metody, moc by nám Maple neposloužil. Mathematica nám dává možnost řešit nelineární rovnice s možností dobré nastavitelnosti. Uživatel má na výběr ze 3 implementovaných metod. Náš program dokáže řešit jak polynomiální rovnice, tak transcendentní rovnice metodami základními i modifikovanými. Řeší rovnice s velkou možností nastavitelnosti a je to nástroj pro efektivní porovnávání metod v různých situacích.

2. Představení matematického problému

Touto kapitolou se pokusíme uvést čtenáře do našeho problému. Zformulujeme matematickou úlohu hledání kořene nelineární rovnice z hlediska numerické matematiky. Uvedeme si nejužívanější metody výpočtu řešení nelineární rovnice. V neposlední řadě zformulujeme pár vět, které se týkají již zmíněných metod řešení rovnic. Půjde nám především o výklad základních metod, s nastíněním problému, které se vážou s jejich použitím. Po vytknutí hlavních problémů navrhneme možnosti, jak tyto nedostatky řešit.

2.1. Úvodní pojmy

Definice: *Nelineární rovnicí* rozumíme

(2.1) $$f(x) = 0,$$

kde *f* je funkce, která bude splňovat jisté předpoklady.

Definice: *Iterační metody* pro řešení rovnice (2.1) jsou takové metody, které nám dávají posloupnost konvergující k řešení rovnice (2.1).

V základě bychom mohli iterační metody rozdělit do dvou skupin. První skupinu bychom mohli nazvat *vždy konvergentní*. Jak již název napovídá, výpočet bude konvergovat pro každou spojitou funkci *f* a to pro jakékoliv počáteční parametry. Avšak u vždy konvergentních metod je problém, že většinou konvergují pomalu. Na druhou stranu mluvíme o metodách, u nichž předpoklady pro konvergenci jsou velmi málo omezující jak na funkci *f*, tak na volbu počátečního přiblížení.

Druhou skupinu metod tvoří *rychle konvergentní* metody, o nichž je známo, že konvergují rychleji něž vždy konvergentní metody. Předpoklady pro konvergenci jsou však o dost více omezující něž u vždy konvergentních metod.

V *úloze hledání řešení* rovnice (2.1) se snažíme nalézt takové x^*, pro které by platilo

$$f(x^*) = 0.$$

Neboli cílem je generovat posloupnost $\{x^{(k)}\}$ takovou, že $\lim_{n\to\infty} x^{(k)} = x^*$, kde $f(x^*) = 0$.

2.2. Základní metody pro řešení nelineárních rovnic

V následujících odstavcích si představíme základní iterační metody pro řešení rovnic. Nepůjde nám o výčet všech metod a jejich modifikací, nýbrž se pokusíme o seznámení s metodami, o vysvětlení toho, jak vlastně fungují, a o nástin problémů, které se váží s jejich používáním.

2.2.1. Jednoduchá metoda

Jednoduchá metoda představuje naivní algoritmus, jak bychom mohli přistupovat k problému. Ačkoliv jednoduchá metoda je pro člověka nejintuitivnější a nejsnadnější pro pochopení, nedá se v praxi použít.

Pro zadanou spojitou funkci f a přesnost p se snažíme najít přibližný kořen x_k takový, aby platilo $|x_k - x^*| < p$, kde x^* je skutečný kořen rovnice. Představme si, že máme interval $[a, b]$. Platí-li pro funkční hodnoty v krajních bodech intervalu

(2.2) $$f(a).f(b) < 0,$$

tak to znamená, že graf funkce přechází z jedné strany osy x na druhou, a rovnice má v tomto intervalu alespoň jeden kořen. Jak tedy takovou aproximaci x_k pro zadanou spojitou funkci *f*, počáteční interval $[a, b]$ a přesnost *p* hledat?

Rozdělíme si interval $[a, b]$ na dílčí intervaly o délce přesnost. Postupně bereme jeden dílčí interval za druhým a testujeme pro něj podmínku (2.2). Když takový interval najdeme, prohlásíme jeho střed za nalezený kořen rovnice.

Jak sami jistě tušíte, efektivnost takového algoritmu je velmi nepostačující. Například pro interval o velikosti 20 a přesnosti 0,0001 by algoritmus mohl vykonat až 200 000 iterací. Určitě bychom si všichni přáli, aby to šlo rychleji.

2.2.2. Metoda bisekce (metoda půlení intervalu)

Tato metoda je nejprostší používaná metoda, která patří mezi vždy konvergentní metody pro řešení rovnice (2.1). Mějme funkci *f(x)* a interval *[a, b]*. Dále mějme tyto předpoklady:

(2.3) *f(x)* je spojitá na $[a, b]$, *f(a) . f(b) < 0*

K určení kořene rovnice (2.1), který leží v intervalu *[a, b]*, budeme postupovat tak, že rozdělíme tento interval na polovinu. Střed intervalu označme písmenem

$s = f\left(\frac{a+b}{2}\right)$. Pokud získáme $f(s) = 0$, pak dostáváme kořen rovnice $x^* = s$. Jestli tomu tak není a $f(s)$ je různé od nuly, vybereme buď levý interval $[a, s]$ nebo $[s, b]$ a provedeme na něm stejný postup jako u předchozího. Interval poloviční délky vybereme tak, aby jeho koncové body měly funkční hodnoty s rozdílnými znaménky. Tímto postupem dojdeme po konečném počtu kroků k přesnému kořenu rovnice x^* a nebo najdeme takovou aproximaci x_k, která vyhovuje našemu požadavku na přesnost. Vyslovme nyní větu o konvergenci metody bisekce.

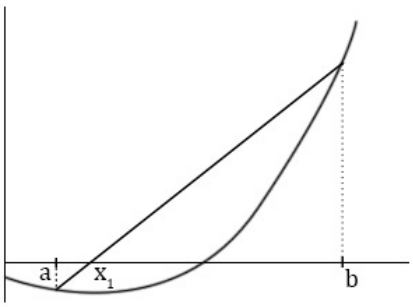
Věta: Buď f je spojitá funkce na intervalu $[a, b]$ a platí $f(a) \cdot f(b) < 0$. Dále nechť $\{x_n\}_{n=0}^{\infty}$ je posloupnost, kterou dostaneme metodou bisekce. Pak tato posloupnost konverguje k $\propto$, kde $f(\propto) = 0$.

Důkaz této věty si zde nebudeme uvádět. Pokud by čtenáře zajímal formální matematický důkaz, může ho najít například v knize Základy numerické matematiky od B. P. Děmidoviče a I. A. Marona [2].

Metoda bisekce se prakticky hodí jen k velmi přibližnému určení kořene. Při zvětšováni přesnosti velmi vzrůstá počet iterací a výpočet se nám může dosti protáhnout.

2.2.3. Metoda regula falsi (metoda sečen)

Ukažme si nyní rychlejší metodu oproti metodě bisekce. Platí pro ni stejné předpoklady (2.3). Jedná se o metodu regula falsi (metoda sečen nebo také metoda tětiv). Oproti metodě bisekce nedělíme počáteční interval $[a, b]$ na polovinu. Je přirozenější interval dělit v poměru $|f(a)| : |f(b)|$. Rozdělením intervalu dostaneme bod x_k, který nazveme průsečíkem (hned si vysvětlíme proč). Graficky bychom si to mohli představit jako na obrázku 2.1:



Obr. 2.1 - metoda regula falsi

Jak jde z obrázku 2.1 vidět, sestrojíme tětivu (sečnu) mezi body A a B. Vzniklý bod x_1 je průsečík tětivy a osy x. Číselně můžeme vypočítat první průsečík x_1 z následujícího vzorce:

$$x_1 = \frac{a\,f(b) - b\,f(a)}{f(b) - f(a)}$$

Poznámka: Výše uvedený vzorec je jednoduše vyjádření bodu průsečíku sečny a osy x v analytické geometrii.

Tento postup posléze opakujeme pro ten z intervalů $[a, x_1]$ a $[x_1, b]$, na jehož koncích mají hodnoty funkce *f(x)* opačná znaménka.

Metoda regula falsi patří mezi vždy konvergentní metody stejně jako metoda bisekce. Na řadu nyní přichází vyslovit větu o konvergenci metody regula falsi.

Věta: Buď *f* je spojitá funkce na intervalu *[a, b]* a platí *f(a) . f(b) < 0*. Dále nechť $\{x_n\}_{n=0}^{\infty}$ je posloupnost, kterou dostaneme užitím metody regula falsi. Pak tato posloupnost konverguje k $\propto$, kde $f(\propto) = 0$.

Čtenáře odkážeme na knihu Základy numerické matematiky od J. Segethové [3], , pokud by ho zajímal důkaz této věty, jelikož se jím zde zabývat nebudeme.

V obecném případě je metoda regula falsi rychlejší než metoda bisekce, ale jsou výjimky, kdy to neplatí. Čtenář si později za použití programu může zkusit spočítat rovnici $e^x - 2 = 0$ na intervalu [*-10, 10*].

Metoda bisekce a metoda regula falsi jsou příklady dvou nejběžnějších vždy konvergentních metod. Jsou ale pomalé a v praxi se používají jen k nalezení přibližného řešení pro další výpočet.

2.2.4. Newtonova metoda (metoda tečen)

Newtonova metody (neboli metody tečen) patří mezí rychle konvergentní metody pro řešení rovnice (2.1). Může však být problematická, protože nekonverguje vždy, jako jsme byli zvyklí u metody bisekce nebo u metody regula falsi.

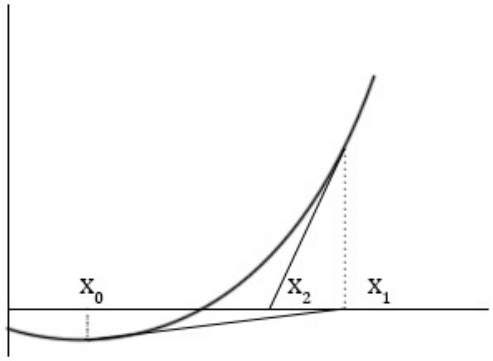
Myšlenka Newtonovy metody je opět vcelku jednoduchá. Nezačínáme s počátečním intervalem, ale s přibližným kořenem x_0 (přibližnou aproximací). Z bodu $[x_0, f(x_0)]$ sestrojíme tečnu ke grafu funkce (proto metoda tečen) a vzniklý průsečík tečny a

osy x bude naše nová aproximace x_1. Obecně bychom mohli výpočet nové aproximace vyjádřit vzorcem:

(2.4)

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\,, n = 0, 1, \ldots$$

za předpokladu, že derivace $f'(x_n)$ je nenulová pro každé *n*. Popisovanou situaci znázorňuje následující obrázek 2.2:



Obr. 2.2 - Newtonova metoda

Newtonova metoda však není vždy konvergentní. Pokud se nám poštěstí a my zvolíme dostatečně blízkou aproximaci kořene, Newtonova metoda najde kořen rychle. Pro konvergenci Newtonovy metody platí následující věta:

Věta: Nechť $f \in C^2[a,b]$, $f(a)f(b) < 0$, $f'(x) \neq 0$ pro všechna $x \in [a,b]$ a nechť f'' nezmění znaménko v intervalu $[a,\ b]$. Pak zvolme $x_0 \in [a,b]$ takové, že $f(x_0)f''(x_0) > 0$. Pak posloupnost $\{x_n\}_{n=0}^{\infty}$ vzniklá Newtonovou metodou konverguje k bodu $\propto$, kde $f(\propto) = 0$.

Má-li čtenář zájem o důkaz této věty, odkážeme ho opět na knihu [3], kde je důkaz podrobně rozebírán.

2.2.5. Monitorování funkce

Uvažme funkci *f(x)*, která nesplňuje podmínky pro konvergenci Newtonovy metody. I když funkce nesplňuje podmínky konvergence, tak se pokusíme rovnici řešit. Jak zjistit v dané iteraci výpočtu Newtonovy metody, jestli má smysl pokračovat dál nebo jestli další výpočet nemá smysl? Pro tento účel se používá tzv. *monitorování funkce* (z anglického function monitoring). Jedná se o způsob, jak se po dané iteraci

rozhodnout, jestli budeme s výpočtem pokračovat nebo jestli ukončíme výpočet a prohlásíme, že Newtonova metoda nemohla najít za daných parametrů nalézt žádný kořen.

Jeden ze způsobů, jak provádět monitorování funkce, je, že si budeme počítat vzdálenosti aproximací. V nejnovější iteraci jsme dostali aproximaci x_{k+1} a spočteme si pro ni vzdálenost $|x_{k+1} - x_k|$. Tu porovnáme se vzdáleností z předešlé iterace, tj. $|x_k - x_{k-1}|$ a pokud platí

$$|x_{k+1} - x_k| < |x_k - x_{k-1}|$$

pak budeme s výpočtem pokračovat, jinak výpočet ukončíme. Je to jakýsi způsob jak zabudovat podmínku konvergence přímo do algoritmu výpočtu ve formě zkracování vzdáleností aproximací. Tento způsob monitorování je implementovaný v základní nemodifikované Newtonově metodě (viz kapitola 3.1.2., část NewtonsMethod).

Další způsob monitorování funkce můžeme realizovat tak, že budeme požadovat, aby posloupnost funkčních hodnot aproximací konvergoval k 0. Máme podmínku

(2.5)

$$\left|\frac{f(x_{k+1})}{f(x_k)}\right| < 1$$

Pokud je podmínka (2.5) splněna, pak s výpočtem pokračujeme, jinak ne. Koeficient 1 v podmínce (2.5) není pevně daný. Jak je uvedeno v knize [4], je možné používat například i koeficienty 0,5 nebo 0,9. Není ani vyloučeno, že by výpočet nemohl být úspěšný, kdybychom zvolili koeficient 1,2. Čím menší koeficient, tím přísnější máme podmínku. V Newtonově damping metodě, která je popsána v kapitole 2.3.2. a implementovaná v našem programu, jsme použili však koeficient 1.

Čtenář si nyní může myslet, že tyto podmínky monitorování funkce nám přeci nezaručují konvergenci. A taky že má pravdu. Celý problém je ale v Newtonově metodě. Neexistuje jiná věta, která by dokazovala konvergenci Newtonovy metody než ta věta se velmi přísnými podmínkami, kterou jsme zde uváděli. Tedy když počítáme rovnici, která nám nesplňuje přísné podmínky konvergence pro Newtonovu metodu, nezbývá nám nic jiného něž počítat naslepo a doufat, že dojdeme k nějakému výsledku.

2.3. Nedostatky základních metod a jejich možná řešení

Jak si můžeme povšimnout, podmínky pro konvergenci Newtonovy metody jsou velmi přísné a ne vždy je můžeme splnit. Nedalo by nám ani moc práce, abychom našli případ, aby se Newtonova metoda pokazila. Oproti rychle konvergující Newtonově metodě metoda bisekce a metoda regula falsi konvergují pomalu. Co dělat, kdybychom chtěli mít oboje, tedy metodu, která neklade velká nároky pro konvergenci a přitom konverguje rychle? V níže psaných odstavcích si popíšeme 2 alternativy, jak se s tímto problémem dá vypořádat.

2.3.1. Newtonova hybridní metoda

Jak jsme si již řekli, Newtonova metoda při špatně zvoleném počátečním řešení konverguje k řešení rovnice (2.1) velmi pomalu nebo dokonce vůbec. V knihách [2] a [3] je uvedeno, že se metoda bisekce a metoda regula falsi používá pro nalezení přibližného kořene a posléze se výpočet přepne na nějakou jinou metodu rychle konvergující. Pokud spojíme metodu bisekce nebo metodu regula falsi s Newtonovou metodou, vznikne nám tzv. *hybridní metoda*. V praxi se nechává proběhnout pár iterací metody bisekce nebo metody regula falsi, aby se dostatečně přiblížilo ke skutečnému kořenu rovnice, a zbytek se dopočítá Newtonovou metodou.

Co kdybychom ale zašli ještě dále? Víme, že Newtonova metoda v dobrém případě konverguje ke kořenu rovnice rychleji než metoda bisekce nebo metoda regula falsi. Problém je, že se v některé iteraci může pokazit a neprojde nám podmínka monitorování funkce. Zkusme tedy výpočet provést tak, že když narazíme na problém, přepneme se do "bezpečného módu" a provedeme tolik iterací vždy konvergentní metody, dokud nebudeme zase moct nasadit rychle konvergentní metodu (jak je vysvětlováno třeba ve článku [21]).

Výpočet hybridní metody započneme Newtonovou metodou. V každé iteraci provedeme monitorování funkce. Pokud uspějeme, pokračujeme dál ve výpočtu Newtonovou metodou. Ve chvíli, kdy zmonitorujeme funkci s negativním výsledkem, tak přepneme třeba na metodu regula falsi. Provedeme tolik iterací metody regula falsi, dokud nezmonitorujeme funkci s pozitivním výsledkem. Pak opět přepneme na Newtonovu metodu.

Hybridní metodě neklademe na začátku žádné podmínky. Metoda regula falsi a metoda bisekce vyžadují, aby funkční hodnoty na koncích intervalu měly opačná znaménka. Pro provedení iterace Newtonovou metodou to však nepotřebujeme. Zkoušíme tedy iterovat Newtonovou metodou, dokud to jde. Pokud se dostaneme do situace, kdy už to nejde, tak v dané iterací vyžadujeme podmínku *f(a) . f(b) < 0,* která je esenciální pro provedení iterace metody regula falsi nebo metody bisekce. Pokud v dané chvíli ta podmínka splněna není, tak ukončíme výpočet a prohlásíme, že hybridní metoda za daných vstupních parametrů není schopna nalézt kořen rovnice.

Newtonova metoda nevyžaduje jako metoda regula falsi nebo metoda bisekce počáteční interval. Newtonova metoda potřebuje počáteční kořen. Když tedy dostaneme vstupní interval [a, b], zvolíme jako počáteční řešení střed intervalu [a, b].

Jak může tedy čtenář vycítit, hybridní Newtonova metoda je navržena tak, aby se zakrývaly nedostatky jednotlivých metod. Zkombinováním metody rychle konvergentní a metody vždy konvergentní vznikla metoda schopnější efektivnějšího výkonu.

2.3.2. Newtonova damping metoda

Vzpomeňme na iterační vzorec pro Newtonovu metodu (2.4). Části vzorce (2.4)

$$\frac{f(x_n)}{f'(x_n)} = d_n$$

se často říká *diference*. Při výpočtu klasické Newtonovy metody dochází mnohdy k fenoménu zvaný *"přestřelení"* (z anglického overshoot).

V knize Newton Methods for Nonlinear Problems [4] od P. Deuflharda je představena metoda, která se nazývá *Newtonova damping metoda*. Jedná se o modifikaci algoritmu Newtonovy metody, kde zavádíme jistý *damping koeficient* $\propto \in (0, 1]$ (často také nazýván Deuflhardův damping faktor). A právě tento damping koeficient je určený k usměrnění výpočtu v případě "přestřelení".

Vzorec pro výpočet nové iterace upravíme následovně:

$$x_{n+1} = x_n - \propto \frac{f(x_n)}{f'(x_n)}\ , n = 0, 1, \ldots$$

Damping koeficient se používá tak, že na začátku je nastaven $\propto = 1$ (tedy klasická Newtonova metoda). Ve chvíli, kdy nastane "přestřelení" a neprojde nám podmínka monitorování funkce, začneme upravovat damping koeficient $\propto$. V knize [4] je popisován další postup tak, že damping koeficient nastavíme na polovinu

$$\propto := \frac{\propto}{2}$$

a opět zkusíme zmonitorovat funkci. Damping koeficient půlíme tak dlouho, dokud nám neprojde monitorování funkce. Tomuto procesu se říká *damping*. Pokud nám ani po několika úpravách diference se nedaří projít monitorováním funkce, tak nemá cenu pokračovat dál a ukončíme výpočet. Má cenu půlit damping koeficient maximálně desetkrát (viz kniha [4], kapitola 3). Pokud ani tehdy nám to nic nepřineslo, nemá cenu dál pokračovat. Pokud ale nalezneme takový damping koeficient, aby nám prošla nová iterace na monitorování funkce, tak pokračujeme dál a v nové iteraci bereme opět $\propto = 1$.

Poznámka: V kapitole 3 knihy [4] jsou popisovány strategie, jak upravit damping koeficient pro další iteraci za stávajících hodnot v nynější iteraci. Avšak my se tímto zde zabývat nebudeme, ale kdyby čtenáře zajímala další možnost pro variaci výpočtu, nechť se odkáže na knihu [4].

3. Programátorská dokumentace

Třetí kapitolou této práce představíme čtenáři náš program z vnitřního pohledu. Rozebereme si program z nejsvrchnější vrstvy a pomalu se budeme dostávat hlouběji a detailněji k jednotlivým částem kódu programu. Po přečtení této kapitoly by čtenář měl mít představu o tom, jak byl program navrhován, jak byly implementovány jednotlivé části programu a jak spolu různé části souvisejí. Nejde nám o výklad algoritmů a postupů. Tyto věci se dají vyčíst ze zdrojového kódu. Z této kapitoly by právě měl čtenář zjistit to, co ve zdrojovém kódu nemohl.

Náš program je stavěný na platformě .NET Framework [13] od Microsoftu [14]. Pro jeho spuštění je třeba na počítači mít nainstalovaný .NET Framework verze 3.5 a vyšší.

Struktura celého programu by se dala rozdělit na dvě vrstvy. Jedná se o vrstvu prezenční a vrstvu výpočetní. *Výpočetní vrstva* nějak nakládá s daty, které dostal program od uživatele. Podle navolených parametrů a zadaných dat se pokusí program provést výpočet. *Prezenční vrstva* stojí mezi uživatelem a vnitřní logikou programu. Umožňuje uživateli zadávat data a nastavovat parametry výpočtu. V podstatě uživatel pomocí prezenční vrstvy říká programu, co po něm vlastně chce. Zabývejme se napřed vrstvou výpočetní.

3.1. Výpočetní vrstva

Výpočetní vrstvu bychom mohli rozdělit do několika základních části. Výpočetní prvky programu se skládají ze dvou knihoven a početních objektů v hlavním programu. Hlavní program využívá knihoven `ArithmeticLibrary` a `GraphicLibrary`. Nejpodstatnějším početním objektem hlavního programu je pak `EquationSolver` a od něj děděné objekty.

Důvod pro vytvoření dvou knihoven je zřejmý. Knihovny umí takové věci, které využíváme na více místech. Jsou lehce přenositelné a při změně nějaké funkcionality se provede změna jenom na jednom místě a to přímo v kódu knihovny. Kdežto objekty hlavního programu vykonávají přesně to, co má vykonávat samotný program. Jsou tedy silně spjaty s hlavním programem. Probereme si napřed jednotlivé knihovny a pak se dostaneme k objektům výpočetní vrstvy ležící v hlavním programu.

3.1.1. ArithmeticLibrary

Jak již název `AritmheticLibrary` napovídá, jedná se o knihovnu, která má za úkol vykonávat početní operace. Kromě sčítání, odečítání a jiných matematických úkonů umí tato knihovna "parsovat" aritmetický výraz zadaný textovou formou a převést ho do formy, ve které se bude výraz lehce vyhodnocovat. Dále budeme pro naše řešení rovnic numerickými metodami potřebovat nějak reprezentovat nejen funkce, ale i derivace funkcí. To vše obstarává právě knihovna `ArithmeticLibrary`.

Nejrozumnější způsob, jak reprezentovat aritmetický výraz v objektovém programování, je postavit si *aritmetický strom* (jak jsme se mohli dozvědět na přednáškách [15]). Každý uzel stromu reprezentujeme abstraktní třídou `Node`, která svazuje a zastřešuje všechny uzly aritmetického stromu do jednoho. V aritmetickém výraze pak budeme nacházet binární operátory, unární operátory, proměnné a číselné hodnoty. Odpovídají jim třídy `BinaryOperatorNode`, `UnaryOperatorNode`, `VariableNode` a `NumberNode`. Od tříd `BinaryOperatorNode` a `UnaryOperatorNode` pak ještě dědí třídy jednotlivých operátorů, které zde ale rozepisovat nebudeme. Podotkneme pouze, že naše knihovna podporuje sčítání, odečítání, násobení, dělení a mocnit. Z unárních operací umí zpracovat a vyhodnotit unární mínus, goniometrické funkce (sinus, cosinus, tangens), přirozený logaritmus a exponenciální funkci o základu *e*. Měli bychom také ještě podotknout, že aritmetický strom je `immutable`. Znamená to, že ho po vytvoření nemůžeme nijak změnit. Hlavním důvodem pro `immutable` strom je, že můžeme bez problému paralelně volat vyhodnocování na tom samém stromě pro různé hodnoty bez jakékoliv synchronizace.

Pojďme se nyní podívat na třídu `Expression`, která reprezentuje samotný aritmetický výraz. Obsahuje v sobě právě aritmetický strom (pamatuje si pouze referenci na kořen stromu). Třída `Expression` se stará o "parsování" vstupního řetězce a následně řetězec zpracuje a postaví z něj aritmetický strom. Postupuje při tom tak, že napřed výraz v infixu [16] převede na výraz v postfixu [17] (podle algoritmu přednášený na přednášce [15]). Když pak má výraz v postfixu, průchodem od konce už lze lehce rekurzivně stavět strom. Hlavní předností výrazového stromu je to, že všechny operace na stromě můžeme provádět rekurzivně. To je také důvod, proč si třída `Expression` musí pamatovat pouze odkaz na kořen stromu.

Třída `Function` reprezentuje funkci. Mezi jejími položkami najdeme objekty třídy `Expression`, které představují samotnou funkci, její první a druhou derivaci. Když pak potřebujeme vypočítat hodnotu funkce v bodě x nebo hodnotu nějaké derivace v bodě x, zavoláme příslušnou vyhodnocovací metodu a předáme hodnotu x parametrem.

Jak si již čtenář mohl povšimnout, děláme na aritmetickém stromě spoustu operací. Chceme ho jednak vyhodnocovat, ale také ho potřebujeme upravovat v rámci derivování nebo ho chceme vytisknout. Všechny tyto operace bychom mohli implementovat přímo, ale výsledkem by byl velmi nepřehledný kód. Navíc když uvážíme strukturu aritmetického stromu, který se skládá z mnoha různých uzlů jiných typů, tak bychom museli pro každý uzel psát zvlášť odlišnou implementaci. Z tohoto důvodu se při psaní této knihovny využil návrhový vzor *Visitor* [18], který má pomoct právě v této situaci.

Jednotlivé uzly stromu tedy implementují pouze jednu metodu a to je metoda `Visit()`, která pustí určitého visitora dovnitř, a ten visitor pak vykoná práci, ke které je určen. V knihovně najdeme třídu `DerivatorVisitor`, která se stará o přestavění aritmetického stromu v rámci derivace výrazu. Dále tam najdeme třídu `ReducerVisitor`, který má na starosti zjednodušení výrazu o zbytečné části. Pak se tam také nachází třída `InfixPriorityPrinterVisitor`, který se stará o tisk výrazu.

3.1.2. GraphicLibrary

Knihovna `GraphicLibrary` se stará o vykreslování funkcí na obrazovku. Využívá při tom knihovny `ArithmeticLibrary`, kde je implementované počítání s funkcemi a především vyhodnocování funkcí, které budeme potřebovat.

Hlavní třída knihovny je `FunctionDrawer`. Ta získává vstupní data z hlavního programu. V hlavním programu se řeší rovnice a pro výpočet rovnice musel uživatel zadat nějaký interval, na kterém se rovnice řeší. Při vytvoření instance třídy `FunctionDrawer` se předá konstruktoru třídy právě ten interval a my na něm budeme naši funkci vykreslovat. V hlavním programu, kde se nachází vytvořená instance třídy `FunctionDrawer`, obstarává vykreslování objekt typu `Graphics` a vykreslování se provádí do komponenty `Panel`. Objekt `Panel` i `Graphics` se předají

spolu s intervalem konstruktoru třídy `FunctionDrawer`, aby měla všechno potřebné pro vykreslování.

Poznámka: Nyní je v knihovně implementováno kreslení do `Panelu`, ale dá se jednoduše upravit, aby se kreslilo na jinou komponentu.

Samotné vykreslení už není nic složitého. Nejdříve si potřebujeme spočítat tabulku funkčních hodnot (funkčně vyhodnotíme body v zadaném intervalu). Pro každou hodnotu v tabulce pak spočteme její bod, který bude reprezentovat její umístění na `Panelu`. Když jsme počítali tabulku funkčních hodnot, zapamatovali jsme si při tom maximální a minimální funkční hodnotu, která se zde vyskytla. Toto maximum a minimum pak použijeme k tomu, abychom se rozhodli, jaké měřítko na ose y máme zvolit. Poté stačí jen zavolat metodu `Graphics.DrawCurve()`, které předáme jako parametr pole s body, které se mají vykreslit.

3.1.2. Výpočetní objekty v hlavním programu

Výpočetní vrstvu v hlavním programu tvoří třída `EquationSolver`. Tato abstraktní třída reprezentuje obecně objekt, který vykonává samotný výpočet rovnice. Třída `EquationSolver` zastřešuje všechny jednotlivé metody pro řešení nelineárních rovnic. Taktéž objekt typu `EquationSolver` obsahuje všechny datové položky potřebné pro průběh výpočtu. V našem programu jsou implementovány všechny numerické metody, které byly představeny v kapitole 2. Jedná se o jednoduchou metodu, metodu bisekce, metodu regula falsi, Newtonovu metodu, Newtonovu hybridní metodu a Newtonovu damping metodu. Těmto metodám odpovídají následující třídy: `SimpleMethod`, `BisectionMethod`, `SecantMethod`, `NewtonsMethod`, `NewtonsHybridMethod` a `NewtonsDampingMethod`.

Při vytvoření instance jakékoliv metody pro řešení rovnice se předá třídě konstruktorem všechna potřebná vstupní data ze třídy `MainForm`, která spadá do prezenční vrstvy. `MainForm` totiž obsahuje všechny parametry a vstupní data, která zadal uživatel. Mluvíme o zadané funkci, intervalu, na kterém se má řešit rovnice, počátečním kořenu, přesnosti výpočtu a maximálním počtu iterací.

Třídy odvozené od třídy `EquationSolver` mají předepsáno, že musí implementovat metodu `Solve()`, která obsahuje kód algoritmu každé metody. Dále každá třída metody pro řešení rovnice obsahují položky `start`, `end` a `precision`. Jedná se

o začátek a konec intervalu, na kterém má výpočet probíhat, a přesnost, se kterou má program počítat. Výjimkou jsou třídy `NewtonsMethod` a `NewtonsDampingMethod`. Ty pro řešení rovnice nepotřebují interval, ale počátečný přibližný kořen. Podívejme se nyní na jednotlivé metody.

SimpleMethod, BisectionMethod, SecantMethod

Metody `Solve()` tříd `SimpleMethod`, `BisectionMethod` a `SecantMethod` jsou dosti přímočaré. Provádí výpočet přesně podle algoritmu popsaný v kapitolách 2.2.1., 2.2.2. a 2.2.3. V každé metodě je hlavní cyklus, který iteruje tak dlouho, dokud nenajde kořen nebo dokud nepřekročí limit iterací, který byl zadán uživatelem. U metod `BisectionMethod.Solve()` a `SecantMethod.Solve()` se může také stát, že výpočet nezačne, protože není splněna podmínka vyžadující, aby funkční hodnoty bodů na konci intervalu měly opačná znaménka (viz kapitola 2.2.2. a 2.2.3.). V takovém případě program vyhodí příslušnou výjimku informující, že se vybraná metoda nehodí pro řešení rovnice s danými parametry.

NewtonsMethod

Metoda `Solve()` třídy `NewtonsMethod` realizuje výpočet nemodifikované Newtonovy metody. Algoritmus této metody je popsán v kapitole 2.2.4. Povšimněte si ve zdrojovém kódu proměnných `oldStep` a `newStep`. Toto jsou právě ty proměnné, které se používají pro monitorování funkce metodou krácení přibližných kořenů (viz kapitola 2.2.5.). Stane-li se v nějakém kroku výpočtu, že hodnota proměnné `newStep` bude větší než hodnota `oldStep` (tj. monitorování funkce proběhlo s negativním výsledkem), ukončíme výpočet a vyhodíme výjimku se zprávou, že je třeba zadat jiný počáteční kořen.

NewtonsHybridMethod

Newtonova hybridní metoda, popisovaná v kapitole 2.3.1., je implementovaná v metodě `Solve()` třídy `NewtonsHybridMethod`. Pozastavme se na chvíli u této třídy. Hybridní metoda využívá jiných metod (metoda bisekce, metoda regula falsi, Newtonova metoda). Na první pohled nás to svádí k tomu, abychom vytvořili ve třídě `NewtonsHybridMethod` jednotlivé instace tříd `BisectionMethod`, `SecantMethod` a `NewtonsMethod` a snažili se nějak využít funkcionalitu, kterou jsme již programovali v jednotlivých metodách. Opak je však pravdou. Třídy

`BisectionMethod`, `SecantMethod` a `NewtonsMethod` jsou odvozené od třídy `EquationSolver`. Tyto třídy reprezentují jakési prostředí, ve kterém se počítá celá rovnice. A když se nad tím zamyslíme hlouběji - to, co vyžadujeme pro třídu `NewtonsHybridMethod` nejsou samotní řešitelé. Potřebujeme pouze něco, co nám provede jednu iteraci nějaké metody. Z tohoto důvodu vznikly třídy `BisectionIterator`, `SecantIterator` a `NewtonsIterator`. Jedná se o třídy, které vykonávají iteraci dané metody a nic víc.

Z důvodu přehlednějšího a čistšího kódu bez zbytečných duplikací využívá třídy `BisectionMethod` iterátor `BisectionIterator` a třída `SecantMethod` iterátor `SecantIterator`. Výjimkou je tomu však u třídy `NewtonsMethod`, která `NewtonsIterator` nevyužívá. Metoda `Solve()` třídy `NewtonsMethod` počítá s počátečním kořenem zadaným uživatelem, kdežto `NewtonsIterator` si vypočte svůj vlastní počáteční kořen jako střed zadaného intervalu. Je tomu tak, protože `NewtonsIterator` je navržený pro práci se třídou `NewtonsHybridMethod`, kde se nepočítá s počátečním kořenem, ale s intervalem.

V metodě `NewtonsHybridMethod.Solve()` je nyní zkombinovaná metoda Newtonova a metoda regula falsi. Kdybychom ale chtěli změnit kombinaci na Newtonovu metodu a metodu bisekce, tak bychom při vytvoření nové instance `NewtonsHybridMethod` konstruktoru předali typ BisectionMethod namísto typu SecantMethod.

NewtonsDampingMethod

Metoda `Solve()` třídy `NewtonsDampingMethod` implementuje algoritmus uvedený v kapitole 2.3.2. za povšimnutí zde stojí metoda `FunctionMonitor()`. Opět jako u nemodifikované Newtonovy metody je třeba provádět nějaké monitorování funkce, abychom mohli říct, zda výpočet někam vede nebo ne. V metodě `Solve()` této třídy jsme implementovali druhý způsob monitorování funkce (viz kapitola 2.2.5.), kde sledujeme, jestli se v nové iteraci funkční hodnota nového přibližného kořene zmenšila nebo ne. Pokud nám v některé iteraci skončilo monitorování funkce s negativním výsledkem, provedeme damping (viz. kapitola 2.3.2.). Pokud se dostaneme pomocí damping na nějakou aproximaci, která nám projde monitorováním funkce, tak pokračujeme ve výpočtu. Pokud damping selže desetkrát

v jedné iteraci, ukončíme výpočet a vyhodíme výjimku, že Newtonova damping metoda nemohla najít za daných parametrů žádný kořen.

Zmiňme se ještě o třídě `MethodFactory`, která se stará o vytváření instancí jednotlivých tříd odvozených od třídy `EquationSolver`. Tato třída byla naprogramovaná podle návrhového vzoru *Factory Method* [19].

3.2. Prezenční vrstva

Prezenční vrstva našeho programu je tvořena z formulářů psané v grafickém rozhraní WinForms [20]. Jedná se tedy o okenní aplikaci. Pomocí formulářů zadává uživatel veškeré vstupní informace programu jako rovnice, metoda, kterou chce uživatel rovnici řešit, a další vstupní parametry potřebné pro výpočet. V našem programu máme jedno hlavní okno `MainForm` a dále 3 menší vedlejší okna `DrawingForm`, `ConfigForm` a `AboutForm`. V dalším textu se jen lehce podíváme dovnitř tříd `MainForm`, `DrawingForm` a `ConfigForm`, nebudeme se zabývat vizuální stránkou a samotným GUI. Těmto věcem se budeme věnovat v kapitole 4.

3.2.1. MainForm

Třída `MainForm` obsahuje spoustu datových položek, ve kterých si pamatuje data, která zadal uživatel. Všechny tyto položky jsou soukromé, aby k nim nemohl nikdo z venčí přistupovat. Ve chvíli, kdy se vytvoří instance nějakého řešitele rovnice, se řešiteli předají konstruktorem všechny datové položky ze třídy `MainForm`, které jsou potřebné pro výpočet.

Tlačítko Nakreslit vytvoří nový `DrawingForm` - okno pro vykreslení zadané funkce. `MainForm` předá konstruktorem `DrawingFormu` funkci, a interval, pokud byly tyto údaje zadány.

Tlačítko Vyřešit zjistí, jaký mód byl zvolen a podle toho provede výpočet. Pokud byla zvolen mód jedné metody, pak podle vlastnosti `Tag` u radio buttonů vyčte, která metoda byla zvolena, a vytvoří pomocí `MethodFactory` příslušný řešitel rovnice. Pokud byl zvolen mód více metod, vytvoří se instance všech řešitelů, které byly zaškrtnuté check boxy. Budeme předpokládat, že výpočet může být časově náročný ať už počítáme jednou metodou nebo více. Z tohoto důvodu jsme v `MainForm` vytvořili `BackgroundWorker`, který bude provádět samotný výpočet všemi zvolenými metodami v odděleném vlákně. Před výpočtem nastavíme tlačítku Vyřešit vlastnost `Enabled` na hodnotu `false`, abychom zamezili vyrušení výpočtu. Ve chvíli, kdy `BackgroundWorker` dokončí svoji práci, znovu zpřístupníme tlačítko vyřešit pomocí vlastnosti `Enabled`. Celá tato věc ohledně `BackgroundWorkeru` a počítání ve vedlejším vlákně je trochu pracnější na implementaci, ale výsledkem toho při dlouhých výpočtech nebude program tuhnout.

3.2.2. DrawingForm

Okno `DrawingForm` slouží k vykreslování funkce. O samotné vykreslování se stará knihovna `GraphicLibrary`. Kreslení se provádí na událost panelu - `Paint`. Tlačítko Nakresli pouze vyvolá na panelu `Refresh()`, aby se panel znovu překreslil. Okno `DrawingForm` se také dá zmenšovat a zvětšovat. Při události `Resize` je opět vyvolané na panelu `Refresh()` pro překreslení funkce.

3.2.3. ConfigForm

`ConfigForm` je formulář pro nastavení vedlejších parametrů programu. Přes toto okno se nastavuje maximální počet iterací, které nesmí řešitel překročit při řešení rovnice. Dále se zde dá nastavit počet desetinných míst v nalezených kořenech.

Toto okno je spojeno s externím konfiguračním souborem, které je implicitně pojmenované "config.cfg". Jedná se o prostý textový soubor, ve kterém je uloženo nastavení. Ukládání nastavení do externího souboru je v této situaci nutností. Bylo by pro uživatele určitě frustrující a velmi nepraktické, kdyby si musel po každém spuštění programu znovu nastavovat svoje nastavení.

Ve chvíli, kdy uživatel otevře okno nastavení, se pokusí `ConfigForm` pomocí metody `LoadConfig()` najít konfigurační soubor a načíst z něho uložené nastavení. Pokud z jakéhokoliv důvodu načíst data nepůjde, ať už kvůli nenalezenému souboru nebo že ho program nemohl přečíst nebo v něm byly uloženy data ve špatném formátu, tak se do okna vypíšou výchozí hodnoty maximálního počtu iterací a počet desetinných míst.

Stejně jako `ConfigForm`, `MainForm` má také svoji metodu `LoadConfig()`. Před každým výpočtem načte nastavení ze souboru config.cfg, pokud to jde. Pokud ne, tak počítá s výchozím nastavením maximálního počtu iterací a počtu desetinných míst.

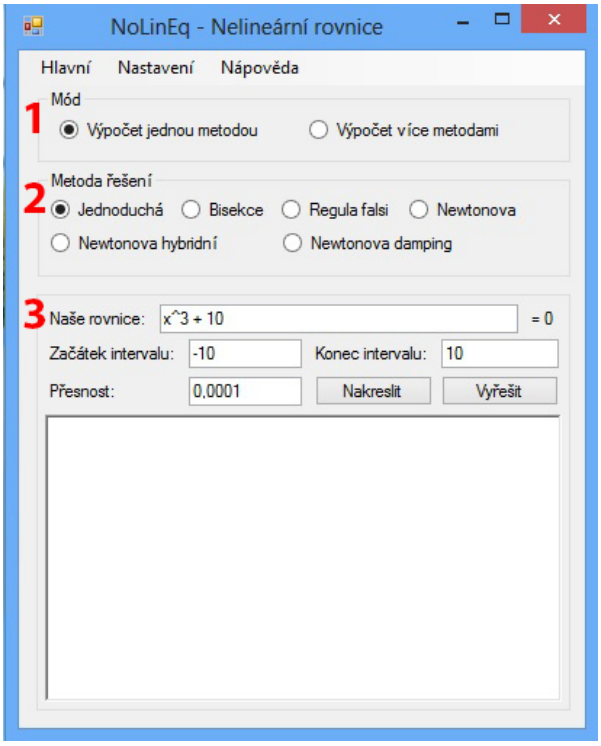
4. Uživatelská dokumentace

Ve čtvrté kapitole seznámíme uživatele s grafickým uživatelským rozhraním našeho programu. Ukážeme si, kam pro co sáhnout a co kde jak nastavit. Také lehce nastíníme možný doporučený postup pro řešení rovnic naším programem.

4.1. Grafické uživatelské rozhraní

V našem programu najdeme několik oken - hlavní okno, okno pro kreslení funkcí, okno nastavení a informační okno.

4.1.1. Hlavní okno

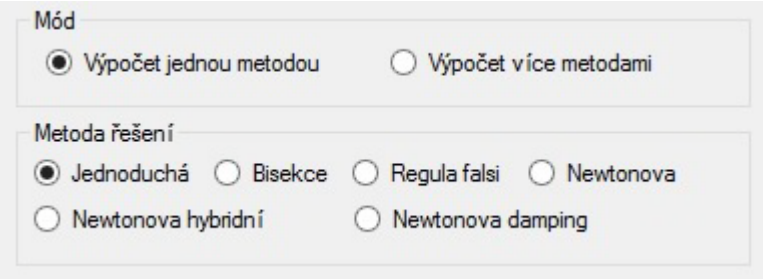


Obr. 4.1 - Hlavní okno

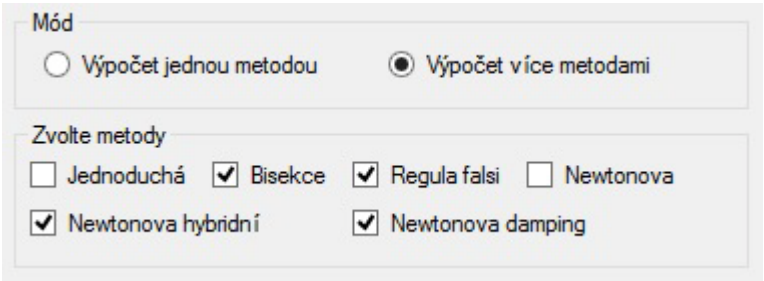
Hlavní okno vidíme na obrázku 4.1. Navrchu se nachází klasické menu. V místě označením číslem (1) je oblast, kde se nastavuje mód programu. Program nabízí počítání ve dvou módech. Uživatel buď může zvolit, že bude testovat jednu konkrétní metodu. Pak zvolí mód výpočtu jednotlivou metodou. Pokud chce testovat více metod na jednom příkladě, zvolí výpočet více metodami.

U čísla (2) je místo, kde se volí metody pro řešení rovnice. Pokud uživatel zvolil mód výpočet jednou metodou, zvolí pak v pomocí kulatého tlačítka, kterou metodu chce

použít (viz obrázek 4.2). Pokud zvolil mód více metod, objeví se zaškrtávací obdélníky a uživatel zaškrtne právě ty metody, kterými chce počítat (viz obrázek 4.3).



Obr. 4.2 - výpočet jednou metodou



Obr. 4.3 - výpočet více metodami

V další části hlavního okna u čísla (3) (viz. obrázek 4.1) je místo, kde se nastavují vstupní data pro výpočet. Uživatel zde může zadat rovnici (zadává se pouze funkce f(x), zadávaná rovnice má být ve tvaru f(x) = 0). Pokud uživatel chce počítat metodou jednoduchou, metodou bisekce, metodou regula falsi nebo metodou hybridní, bude vyzván k zadání intervalu, na kterém se má rovnice řešit. Pokud byla vybrána pro výpočet Newtonova metoda nebo Newtonova damping metoda, pak bude uživatel vyzván k zadání počátečního přibližného kořene. Poté uživatel ještě musí zadat přesnost pro výpočet. Hodnota přesnosti musí být číslo větší než 0 a menší než 0,1. Doporučuje se zadat přesnost 0,0001 a menší.

Poznámka: Newtonova metoda a Newtonova damping metoda vyžadují pro výpočet počáteční kořen a ne interval. Když uživatel zvolí mód výpočtu více metodami a bude mít zvolené i Newtonovu metodu nebo Newtonovu damping metodu, nebude vyzván k zadání počátečního kořene. Program si sám nastaví počáteční kořen jako střed zadaného intervalu.

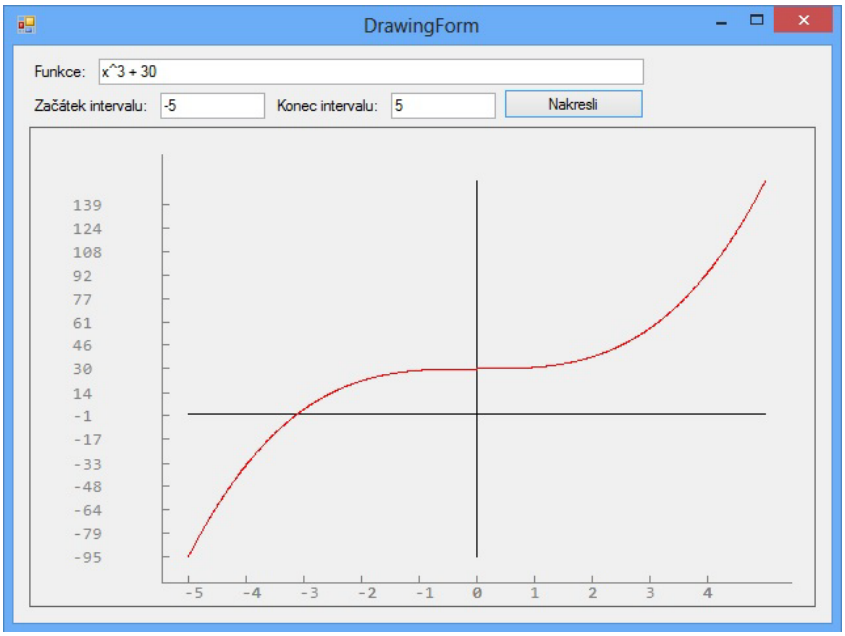
Dále v hlavním okně najdeme tlačítka "Nakreslit" a "Vyřešit". Stiskem tlačítka Nakreslit se vyvolá okno pro kreslení. Tlačítkem vyřešit započne program výpočet.

Nakonec je v hlavním okně dole velké textové pole (log), kam se budou vypisovat výsledky výpočtů.

V horním menu má uživatel možnost zadat nový výpočet. Tím se vymažou veškeré výsledky, které byly dočasně uložené v logu. V případě, že by uživatel chtěl uložit výsledky výpočtů, může tak učinit v menu tlačítkem "Uložit log". Obsah logu se mu uloží do textového souboru pod jménem dnešního data a času vygenerování souboru. Dále v horním menu najde tlačítko Nastavení, kterým se vyvolá okno nastavení programu, a tlačítko "O programu", kde jsou napsány nějaké informace o verzi programu.

4.1.2. Okno kreslení

Okno kreslení se stará o vykreslování zadaných funkcí. Na okno vykreslování se můžeme podívat na obrázku 4.4. Najdeme zde pole pro zadání funkce a intervalu, na kterém se funkce má nakreslit.



Obr. 4.4 - okno kreslení

Pokud uživatel byl uprostřed výpočtu a v hlavním okně měl zadanou nějakou funkci a interval, okno pro vykreslení automaticky zadanou funkci nakreslí.

4.1.3. Okno nastavení

V okně nastavení může uživatel nastavit maximální počet iterací, který nesmí program překročit pro řešení rovnice. Dále se zde může nastavit kolik desetinných míst se má zobrazit ve vypočtených kořenech rovnice.

4.2. Doporučený postup práce s programem

Uživatel se nejdříve rozhodne, jestli chce počítat rovnici jednou metodou nebo více metodami a podle toho si zvolí mód programu. Poté si navolí metody pro počítání. V dalším kroku zadá rovnici, kterou chce řešit.

Aby měl uživatel představu o tom, jaký interval zadat pro řešení, doporučujeme napřed, aby si nechal funkci vykreslit. Když bude mít graf funkce před sebou, dozví se z něj, jestli zadaná funkce v daném intervalu má vůbec nějaký průsečík s osou x (tj. jestli má rovnice v daném intervalů nějaký kořen). Také je to dobré pro ověření, jestli je zadaná funkce v daném intervalu spojitá (spojitost funkce je nutnou podmínkou pro řešení rovnice pro všechny metody, viz kapitola 2) nebo jestli jsou v zadaném intervalu body, které nemají definovanou funkční hodnotu.

Graf funkce také pomůže uživateli pochopit případné chybové hlášky, které se můžou objevit, pokud výpočet rovnice nedošel k toužebnému řešení rovnice. Metoda bisekce, metoda regula falsi potřebují mít zadaný takový interval, aby funkční hodnoty bodů na koncích zadaného intervalu měly opačná znaménka. Pomocí nakresleného grafu funkce se tato chyba lehce odhalí.

Další chybová hláška, která může uživatele zaskočit, je chyba nulové derivace v nějakém bodě. Tato chyba se může vyskytnout při užití Newtonovy metody nebo Newtonovy damping metody. Obě metody totiž vyžadují, aby v žádné iteraci nebyla hodnota první derivace nulová (viz kapitola 2.2.4.). Uživatel si tedy bude muset dát pozor na stacionární a inflexní body. Pokud se nastaví počáteční kořen do stacionárního nebo inflexního bodu, výpočet Newtonovou metodou nebo Newtonovou damping metodou bude ukončen.

Pro výpočet se doporučuje nastavit zobrazení alespoň pěti desetinných míst ve výsledcích a přesnost menší než 0,0001.

Doslov / Závěr

Závěrem této bakalářské práce shrňme, jakou problematikou jsme se zabývali, jaké řešení vyznačených problémů jsme navrhli a jak jsme náš návrh řešili. A nakonec zhodnotíme naše úsilí.

Hlavním tématem této práce byly numerické metody pro řešení nelineárních rovnic. K existujícím programům a software jsme se snažili vytvořit nástroj, který by byl schopný počítat rovnice za použití numerických metod jak základními numerickými metodami, tak rozšířenými modifikovanými metodami. Naším hlavním cílem bylo vytvořit program, pomocí kterého by uživatel mohl efektivně studovat různé metody, porovnávat jejich efektivnost a sledovat jejich výkon v různých situacích.

V kapitole 2 jsme uvedli čtenáře do matematického problému. Čtenář se seznámil se základními numerickými metodami, které se používají pro řešení rovnic. Bylo zde poukázáno na rychlostní a funkční nedostatky základních metod a následně jsme navrhli možnosti, jak se s těmito nedostatky vypořádat.

Pro realizaci řešení našeho návrhu jsme vytvořili program, okenní aplikaci založenou na platformě .NET Framework, který je rozdělen do více funkčních jednotek. Ve výsledku jsme získali nástroj, který umí řešit rovnice jednotlivými numerickými metodami zvlášť nebo současně více metodami najednou. Náš program nabízí mnoho možností nastavení pro výpočet, což nabízí uživateli spoustu možností a scénářů, ve kterých může testovat chování různých metod.

V důsledku časového nedostatku jsou v kódu programu malé nedostatky, které se nestačily opravit. Konkrétně bychom mohli zmínit implementaci hybridní metody, kde vnitřní iterátory zasahují a mění datové položky vnější třídy NewtonsHybridMethod. Tomuto jevu zvaný "side efekt" [22] by se mělo vyvarovat. Také se v kódu programu může vyskytnout pár programátorských obratů, které jsou úplně elegantní.

Program v tuto chvíli také nabízí možnosti pro rozšíření. Například by se mohlo lépe zpracovat interpretace výsledků výpočtu více metod. Výsledek se mohl zobrazovat ve sloupcovém grafu. Dále bychom mohli měřit reálná čas výpočtu metod. Další možností pro rozšíření by mohlo být nalezení všech kořenů v zadaném intervalu a jiné.

Reference

[1] Webová dokumentace programu Wolfram Mathematica: http://reference.wolfram.com/mathematica/guide/Mathematica.html

[2] P. B. Děmidovič - I. A. Maron: Základy numerické matematiky. Praha 1966, Státní nakladatelství technické literatury.

[3] J. Segethová: Základy numerické matematiky. Praha 2002, Univerzita Karlova v Prazte - Nakladatelství Kalorinum.

[4] P. Deuflhard: Newton Methods for Nonlinear Problems. 2004, Springers.

[5] Článek z Wikipedie o návrhovém vzoru Dependency Injection: http://en.wikipedia.org/wiki/Dependency_injection

[6] Webová dokumentace programu Maple: http://www.maplesoft.com/support/help/

[7] Webová dokumentace programu Matlab: http://www.mathworks.com/help/

[8] Odkaz na článek z Wikipedie o Brentově metodě pro řešení rovnic: http://en.wikipedia.org/wiki/Brent's_method

[9] J. Borwein - M. P. Skerritt: An Introduction to Modern Mathematical Computing with Mathematica. 2012, Springers.

[10] Antonio Romano: Classical Mechanics with Mathematica. 2012, Birkhauser.

[11] Ing. Vladimír Žák - Matematické výpočty se systémem Maple: http://www.maple.vladimirzak.com/

[12] G. Lindfield - J. Penny: Numerical Methods Using MATLAB. 2012, Academic Press

[13] Odkaz na článek z Wikipedie o platformě .NET Framework: http://cs.wikipedia.org/wiki/.NET

[14] Webové stránky firmy Microsoft: www.microsoft.com

[15] Přednášky Programování I a Programování II přednášené na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze.

[16] Článek z Wikipedie o infixové notaci: http://en.wikipedia.org/wiki/Infix_notation

[17] Článek z Wikipedie o postixové notaci: http://en.wikipedia.org/wiki/Reverse_Polish_notation

[18] Článek z Wikipedie o návrhovém vzoru Visitor: http://cs.wikipedia.org/wiki/Visitor_(návrhový_vzor)

[19] Článek z Wikipedie o návrhovém vzoru Factory Method: http://en.wikipedia.org/wiki/Factory_method_pattern

[20] Webová knihovna MSDN o technologiích .NETu: http://msdn.microsoft.com/en-us/library/

[21] Článek o variacích a hybridních metodách na Newtonovu metodu: http://en.citizendium.org/wiki/Newton's_method#Variations_and_hybrid_methods

[22] Článek z Wikipedie o jevu "side efekt": http://en.wikipedia.org/wiki/Side_effect_(computer_science)